UG10164

i.MX Yocto Project User's Guide

Rev. LF6.6.36_2.1.0 — 30 September 2024 **User guide**

Document information

Information	Content
Keywords	i.MX, Linux, LF6.6.36_2.1.0
Abstract	This document describes how to build an image for an i.MX board by using a Yocto Project build environment. It describes the i.MX release layer and i.MX-specific usage.

1 Overview

This document describes how to build an image for an i.MX board by using a Yocto Project build environment. It describes the i.MX release layer and i.MX-specific usage.

The Yocto Project is an open-source collaboration focused on embedded Linux OS development. For more information on Yocto Project, see the Yocto Project page: www.yoctoproject.org/. There are several documents on the Yocto Project home page that describe in detail how to use the system. To use the basic Yocto Project without the i.MX release layer, follow the instructions in the *Yocto Project Quick Start* found at https://docs.yoctoproject.org/brief-yoctoprojectqs/index.html.

The FSL Yocto Project Community BSP (found at freescale.github.io) is a development community outside NXP providing support for i.MX boards in the Yocto Project environment. i.MX joined the Yocto Project community providing a release based on the Yocto Project framework. Information specific to FSL community BSP use is available on the community web page. This document is an extension of the community BSP documentation.

Files used to build an image are stored in layers. Layers contain different types of customizations and come from different sources. Some of the files in a layer are called recipes. Yocto Project recipes contain the mechanism to retrieve source code, build and package a component. The following lists show the layers used in this release.

i.MX release layer

- `meta-imx`
  - `meta-imx-bsp`: updates for `meta-freescale`, `poky`, and `meta-openembedded` layers
  - `meta-imx-sdk`: updates for `meta-freescale-distros`
  - `meta-imx-ml`: Machine learning recipes
  - `meta-imx-v2x`: V2X recipes only used for i.MX 8DXL
  - `meta-imx-cockpit`: Cockpit recipes for i.MX 8QuadMax

Yocto Project community layers

- `meta-freescale`: Provides support for the base and for i.MX Arm reference boards.
- `meta-freescale-3rdparty`: Provides support for 3rd party and partner boards.
- `meta-freescale-distro`: Additional items to aid in development and exercise board capabilities.
- `fsl-community-bsp-base`: Often renamed to base. Provides base configuration for FSL Community BSP.
- `meta-openembedded`: Collection of layers for the OE-core universe. See layers.openembedded.org/.
- `poky`: Basic Yocto Project items in Poky. See the Poky README for details.
- `meta-browser`: Provides several browsers.
- `meta-qt6`: Provides Qt 6.
- `meta-timesys`: Provides Vigiles tools for monitoring and notification of BSP vulnerabilities (CVEs).

References to community layers in this document are for all the layers in Yocto Project except `meta-imx`. i.MX boards are configured in the `meta-imx` and `meta-freescale` layers. This includes U-Boot, the Linux kernel, and reference board-specific details.

i.MX provides an additional layer called the i.MX BSP Release, named `meta-imx`, to integrate a new i.MX release with the FSL Yocto Project Community BSP. The `meta-imx` layer aims to release the updated and new Yocto Project recipes and machine configurations for new releases that are not yet available on the existing `meta-freescale` and `meta-freescale-distro` layers in the Yocto Project. The contents of the i.MX BSP Release layer are recipes and machine configurations. In many test cases, other layers implement recipes or include files and the i.MX release layer provides updates to the recipes by either appending to a current recipe, or including a component and updating with patches or source locations. Most i.MX release layer recipes are very small because they use what the community has provided and update what is needed for each new package version that is unavailable in the other layers.

The i.MX BSP Release layer also provides image recipes that include all the components needed for a system image to boot, making it easier for the user. Components can be built individually or through an image recipe, which pulls in all the components required in an image into one build process.

The i.MX kernel and U-Boot releases are accessed through i.MX public Git servers. However, several components are released as packages on the i.MX mirror. The package-based recipes pull files from the i.MX mirror instead of a Git location and generate the package needed.

All packages that are released as binary are built with hardware floating point enabled as specified by the DEFAULTTUNE defined in each machine configuration file. Software floating point packages are not provided starting with the jethro releases.

Release LF6.6.36_2.1.0 is released for Yocto Project 5.0 (Scarthgap). The same recipes for Yocto Project 5.0 are going to be upstreamed and made available on the next release of the Yocto Project release. The Yocto Project release cycle lasts roughly six months.

The recipes and patches in `meta-imx` are upstreamed to the community layers. After that is done for a particular component, the files in `meta-imx` are no longer needed and the FSL Yocto Project Community BSP will provide support. The community supports i.MX reference boards, community boards, and third-party boards.

1.1 End user licence agreement

During the setup environment process of the NXP Yocto Project BSP, the NXP End User License Agreement (EULA) is displayed. To continue to use the i.MX Proprietary software, users must agree to the conditions of this license. The agreement to the terms allows the Yocto Project build to untar packages from the i.MX mirror.

Note:

Read this license agreement carefully during the setup process, because once accepted, all further work in the i.MX Yocto Project environment is tied to this accepted agreement.

1.2 References

i.MX has multiple families supported in software. The following are the listed families and SoCs per family. The i.MX Linux Release Notes describes which SoC is supported in the current release. Some previously released SoCs might be buildable in the current release but not validated if they are at the previous validated level.

- i.MX 6 Family: 6QuadPlus, 6Quad, 6DualLite, 6SoloX, 6SLL, 6UltraLite, 6ULL, 6ULZ
- i.MX 7 Family: 7Dual, 7ULP
- i.MX 8 Family: 8QuadMax, 8QuadPlus, 8ULP
- i.MX 8M Family: 8M Plus, 8M Quad, 8M Mini, 8M Nano
- i.MX 8X Family: 8QuadXPlus, 8DXL
- i.MX 9 Family: i.MX 91, i.MX 93, i.MX 95

This release includes the following references and additional information.

- *i.MX Linux Release Notes* (RN00210) - Provides the release information.
- *i.MX Linux User's Guide* (UG10163) - Provides the information on installing U-Boot and Linux OS and using i.MX-specific features.
- *i.MX Yocto Project User's Guide* (UG10164) - Describes the board support package for NXP development systems using Yocto Project to set up host, install tool chain, and build source code to create images.
- *i.MX Porting Guide* (UG10165) - Provides the instructions on porting the BSP to a new board.
- *i.MX Machine Learning User's Guide* (UG10166) - Provides the machine learning information.
- *i.MX DSP User's Guide* (UG10167) - Provides the information on the DSP for i.MX 8.
- *i.MX 8M Plus Camera and Display Guide* (UG10168) - Provides the information on the ISP Independent Sensor Interface API for the i.MX 8M Plus.

- *i.MX Digital Cockpit Hardware Partitioning Enablement for i.MX 8QuadMax* (UG10169) - Provides the i.MX Digital Cockpit hardware solution for i.MX 8QuadMax.
- *i.MX Graphics User's Guide* (UG10159) - Describes the graphics features.
- *Harpoon User's Guide* (HRPNUG_3.1) - Presents the Harpoon release for i.MX 8M device family.
- *i.MX Linux Reference Manual* (RM00293) - Provides the information on Linux drivers for i.MX.
- *i.MX VPU Application Programming Interface Linux Reference Manual* (RM00294) - Provides the reference information on the VPU API on i.MX 6 VPU.
- *EdgeLock Enclave Hardware Security Module API* (RM00284) - This document is a software reference description of the API provided by the i.MX 8ULP, i.MX 93, and i.MX 95 Hardware Security Module (HSM) solutions for the EdgeLock Enclave (ELE) Platform.

The quick start guides contain basic information on the board and setting it up. They are on the NXP website.

- SABRE Platform Quick Start Guide (IMX6QSDPQSG)
- i.MX 6UltraLite EVK Quick Start Guide (IMX6ULTRALITEQSG)
- i.MX 6ULL EVK Quick Start Guide (IMX6ULLQSG)
- i.MX 7Dual SABRE-SD Quick Start Guide (SABRESDBIMX7DUALQSG)
- i.MX 8M Quad Evaluation Kit Quick Start Guide (IMX8MQUADEVKQSG)
- i.MX 8M Mini Evaluation Kit Quick Start Guide (8MMINIEVKQSG)
- i.MX 8M Nano Evaluation Kit Quick Start Guide (8MNANOEVKQSG)
- i.MX 8QuadXPlus Multisensory Enablement Kit Quick Start Guide (IMX8QUADXPLUSQSG)
- i.MX 8QuadMax Multisensory Enablement Kit Quick Start Guide (IMX8QUADMAXQSG)
- i.MX 8M Plus Evaluation Kit Quick Start Guide (IMX8MPLUSQSG)
- i.MX 8ULP EVK Quick Start Guide (IMX8ULPQSG)
- i.MX 8ULP EVK9 Quick Start Guide (IMX8ULPEVK9QSG)
- i.MX 93 EVK Quick Start Guide (IMX93EVKQSG)
- i.MX 93 9x9 QSB Quick Start Guide (93QSBQSG)

Documentation is available online at nxp.com.

- i.MX 6 information is at nxp.com/iMX6series.
- i.MX SABRE information is at nxp.com/imxSABRE.
- i.MX 6UltraLite information is at nxp.com/iMX6UL.
- i.MX 6ULL information is at nxp.com/iMX6ULL.
- i.MX 7Dual information is at nxp.com/iMX7D.
- i.MX 7ULP information is at nxp.com/imx7ulp.
- i.MX 8 information is at nxp.com/imx8.
- i.MX 6ULZ information is at nxp.com/imx6ulz.
- i.MX 93 information is at nxp.com/imx93.
- i.MX 95 information is at nxp.com/imx95.

2 Features

i.MX Yocto Project Release layers have the following features:

- Linux kernel recipe
  - The kernel recipe resides in the `recipes-kernel` folder and integrates an i.MX kernel from the source downloaded from the i.MX Git server. This is done automatically by the recipes in the project.

  - LF6.6.36_2.1.0 is a Linux kernel released for the Yocto Project.
- U-Boot recipe
  - The U-Boot recipe resides in the `recipes-bsp` folder and integrates an i.MX `uboot-imx.git` from the source downloaded from the i.MX Git server.
  - i.MX release LF6.6.36_2.1.0 for the i.MX 6, i.MX 7, i.MX 8, i.MX 93, and i.MX 95 devices uses an updated v2023.04 i.MX U-Boot version. This version has not been updated for all i.MX hardware.
  - The i.MX Yocto Project Community BSP uses `u-boot-fslc` from the mainline, but this is only supported by the U-Boot community and is not supported with the L6.6.36 kernel.
  - The i.MX Yocto Project Community BSP updates the U-Boot versions frequently, so the information above might change as new U-Boot versions are integrated to `meta-freescale` layers and updates from i.MX `u-boot-imx` releases are integrated into the mainline.
- Graphics recipes
  - Graphics recipes reside in `recipes-graphics` folder.
  - Graphics recipes integrate the i.MX graphics package release. For the i.MX boards that have a GPU, the `imx-gpu-viv` recipes package the graphic components for each DISTRO: frame buffer (FB), XWayland, Wayland backend, and Weston compositor (Weston). Only i.MX 6 and i.MX 7 support Frame Buffer.
  - `Xorg-driver` integrates the `xserver-xorg`.
- i.MX package recipes
  `firmware-imx`, `imx-sc-fimrware`, and other packages reside in `recipes-bsp` and pull from the i.MX mirror to build and package into image recipes.
- Multimedia recipes
  - Multimedia recipes reside in recipes-multimedia.
  - Proprietary packages like `imx-codec` and `imx-parser` have recipes pull from the i.MX mirror to build and package into image recipes.
  - Opensource packages have recipes that pull from the public Git Repos on GitHub.
  - Some recipes are provided for codecs that are restricted. Packages for these are not on the i.MX mirror. These packages are available separately. Contact your i.MX Marketing representative to acquire these.
- Core recipes
  Some recipes for rules, such as `udev`, provide updated i.MX rules to be deployed in the system. These recipes are usually updates of policy and are used for customization only. Releases only provide updates if needed.
- Demo recipes
  Demonstration recipes reside in the `meta-imx-sdk` directory. This layer contains image recipes and recipes for customization, such as touch calibration, or recipes for demonstration applications.
- Machine learning recipes
  Machine learning recipes reside in the `meta-imx-ml` directory. This layer contains machine learning recipes for packages like `tensorflow-lite`, `onnx`, and so on.
- Cockpit recipes
  Cockpit recipes reside in `meta-imx-cockpit` and are supported on the i.MX 8QuadMax using the `imx-8qm-cockpit-mek` machine configuration.
  In the layer `meta-nxp-demo-experience`, more demonstration and tool recipes are included. This layer is included in all released full images.

3 Host Setup

To get the Yocto Project expected behavior in a Linux Host Machine, the packages and utilities described below must be installed. An important consideration is the hard disk space required in the host machine. For example, when building on a machine running Ubuntu, the minimum hard disk space required is about 50 GB. It is recommended that at least 120 GB is provided, which is enough to compile all backends together. For building machine learning components, at least 250 GB is recommended.

The recommended minimum Ubuntu version is 20.04 or later. The latest release supports Chromium v91, which requires an increase to the ulimit (number of open files) to 4098.

3.1 Docker

i.MX is now releasing docker setup scripts in imx-docker. Follow the instructions in the readme for setting up a host build machine using docker.

In addition docker on board is enabled with standard manifest by including the `meta-virtualization` layer on i.MX 8 only. This creates a headless system for installing docker containers from external docker hubs.

3.2 Host packages

A Yocto Project build requires specific packages to be installed for the build that are documented under the Yocto Project. Go to Yocto Project Quick Start and check for the packages that must be installed for your build machine.

Essential Yocto Project host packages are:

```
$ sudo apt install gawk wget git diffstat unzip texinfo gcc build-essential
 chrpath socat cpio python3 python3-pip python3-pexpect xz-utils debianutils
 iputils-ping python3-git python3-jinja2 python3-subunit zstd liblz4-tool file
 locales libacl1
```

The configuration tool uses the default version of grep that is on your build machine. If there is a different version of grep in your path, it may cause builds to fail. One workaround is to rename the special version to something not containing "grep".

3.3 Setting up the Repo utility

Repo is a tool built on top of Git that makes it easier to manage projects that contain multiple repositories, which do not need to be on the same server. Repo complements very well the layered nature of the Yocto Project, making it easier for users to add their own layers to the BSP.

To install the “repo” utility, perform these steps:

1. Create a bin folder in the home directory.

```
$ mkdir ~/bin (this step may not be needed if the bin folder already exists)
$ curl https://storage.googleapis.com/git-repo-downloads/repo > ~/bin/repo
$ chmod a+x ~/bin/repo
```

2. Add the following line to the `.bashrc` file to ensure that the `~/bin` folder is in your PATH variable.

```
export PATH=~/bin:$PATH
```

4 Yocto Project Setup

First, make sure that Git is set up properly with the commands below:

```
$ git config --global user.name "Your Name"
$ git config --global user.email "Your Email"
$ git config --list
```

The i.MX Yocto Project BSP Release directory contains a `sources` directory, which contains the recipes used to build one or more build directories, and a set of scripts used to set up the environment.

The recipes used to build the project come from both the community and i.MX. The Yocto Project layers are downloaded to the `sources` directory. This sets up the recipes that are used to build the project.

The following example shows how to download the i.MX Yocto Project Community BSP recipe layers. For this example, a directory called `imx-yocto-bsp` is created for the project. Any name can be used instead of this.

```
$ mkdir imx-yocto-bsp
$ cd imx-yocto-bsp
$ repo init -u https://github.com/nxp-imx/imx-manifest
-b imx-linux-scarthgap -m imx-6.6.36-2.1.0.xml
$ repo sync
```

Note:

https://github.com/nxp-imx/imx-manifest/tree/imx-linux-scarthgap has a list of all manifest files supported in this release.

When this process is completed, the source code is checked out into the directory `imx-yocto-bsp/sources`.

You can perform Repo synchronization, with the command `repo sync`, periodically to update to the latest code.

If errors occur during Repo initialization, try deleting the `.repo` directory and running the Repo initialization command again.

The `repo init` is configured for the latest patches in the line. Follow the instructions in index: imx-manifest.git to retrieve the original GA. Otherwise, GA plus patches are picked up by default. To pick up previous releases from zeus base, add `-m` (release manifest name) at the end of Repo initialization line and it will retrieve previous releases. Examples are provided in the README file in the link provided above.

5 Image Build

This section provides the detailed information along with the process for building an image.

5.1 Build configurations

i.MX provides a script, `imx-setup-release.sh`, that simplifies the setup for i.MX machines. To use the script, the name of the specific machine to be built for needs to be specified as well as the desired graphical backend. The script sets up a directory and the configuration files for the specified machine and backend.

In the `meta-imx` layer, i.MX provides new or updated machine configurations that overlay the `meta-freescale` machine configurations. These files are copied into the `meta-freescale/conf/machine` directory by the `imx-setup-release.sh` script. The following are i.MX machine configuration files that can be selected. Check either the release notes or the machine directory for the latest additions.

- i.MX 6
  - `imx6qpsabresd`
  - `imx6ulevk`
  - `imx6ulz-14x14-evk`
  - `imx6ull14x14evk`
  - `imx6ull9x9evk`
  - `imx6dlsabresd`
  - `imx6qsabresd`
  - `imx6solosabresd`
  - `imx6sxsabresd`

  - imx6sllevk
- i.MX 7
  - imx7dsabresd
  - imx7ulpevk
- i.MX 8
  - imx8qmmek
  - imx8qxpc0mek
  - imx8mqevk
  - imx8mm-lpddr4-evk
  - imx8mm-ddr4-evk
  - imx8mn-lpddr4-evk
  - imx8mn-ddr4-evk
  - imx8mp-lpddr4-evk
  - imx8mp-ddr4-evk
  - imx8dxla1-lpddr4-evk
  - imx8dxlb0-lpddr4-evk
  - imx8dxlb0-ddr3l-evk
  - imx8mnddr3levk
  - imx8ulp-lpddr4-evk
  - imx8ulp-9x9-lpddr4-evk
- i.MX 9
  - imx91-11x11-lpddr4-evk
  - imx93-11x11-lpddr4x-evk
  - imx93-14x14-lpddr4x-evk
  - imx93-9x9-lpddr4-qsb
  - imx95-19x19-lpddr5-evk
  - imx95-15x15-lpddr4x-evk
  - imx95-19x19-verdin

Each build folder must be configured in such way that they only use one distro. Each time the variable `DISTRO_FEATURES` is changed, a clean build folder is needed. Each graphical backend Frame Buffer, Wayland, and XWayland each have a distro configuration. If no DISTRO file is specified, the XWayland distro is set up as default. Distro configurations are saved in the `local.conf` file in the DISTRO setting and are displayed when the bitbake is running. In past releases, we used the poky distro and customized versions and providers in our `layer.conf` but a custom distro is a better solution. When the default poky distro is used, the default community configuration is used. As an i.MX release, we prefer to have a set of configurations that NXP supports and has been testing.

Here are the list of DISTRO configurations. Note that `fsl-imx-fb` is not supported on i.MX 8 and `fsl-imx-x11` is not supported anymore.

- `fsl-imx-wayland`: Pure Wayland graphics.
- `fsl-imx-xwayland`: Wayland graphics and X11. X11 applications using EGL are not supported.
- `fsl-imx-fb`: Frame Buffer graphics - no X11 or Wayland. Frame Buffer is not supported on i.MX 8 and i.MX 9.

Users are welcome to create their own distro file based on one of these to customize their environment without updating the `local.conf` to set preferred versions and providers.

The syntax for the `imx-setup-release.sh` script is shown below:

```
$ DISTRO=<distro name> MACHINE=<machine name> source imx-setup-release.sh -b
 <build dir>
```

`DISTRO=`<distro configuration name> is the distro, which configures the build environment and it is stored in `meta-imx/meta-imx-sdk/conf/distro`.

`MACHINE=`<machine configuration name> is the machine name which points to the configuration file in `conf/machine` in `meta-freescale` and `meta-imx`.

`-b <build dir>` specifies the name of the build directory created by the `imx-setup-release.sh` script.

When the script is run, it prompts the user to accept the EULA. Once the EULA is accepted, the acceptance is stored in `local.conf` inside each build folder and the EULA acceptance query is no longer displayed for that build folder.

After the script runs, the working directory is the one just created by the script, specified with the -b option. A `conf` folder is created containing the files `bblayers.conf` and `local.conf`.

The `<build dir>/conf/bblayers.conf` file contains all the metalayers used in the i.MX Yocto Project release.

The `local.conf` file contains the machine and distro specifications:

```
MACHINE ??= 'imx7ulpevk'
DISTRO ?= 'fsl-imx-xwayland'
ACCEPT_FSL_EULA = "1"
```

The MACHINE configuration can be changed by editing this file, if necessary.

`ACCEPT_FSL_EULA` in the `local.conf` file indicates that you have accepted the conditions of the EULA.

In the `meta-imx` layer, consolidated machine configurations (`imx6qpdlsolox.conf` and `imx6ul7d.conf`) are provided for i.MX 6 and i.MX 7 machines. i.MX uses these to build a common image with all the device trees in one image for testing. Do not use these machines for anything other than testing.

5.2 Choosing an i.MX Yocto project image

The Yocto Project provides some images that are available on different layers. Poky provides some images, `meta-freescale` and `meta-freescale-distro` provide others, and additional image recipes are provided in the `meta-imx` layer. The following table lists various key images, their contents, and the layers that provide the image recipes.

Table 1. i.MX Yocto project images

Image name	Target	Provided by layer
`core-image-minimal`	A small image that only allows a device to boot.	`poky`
`core-image-base`	A console-only image that fully supports the target device hardware.	`poky`
`core-image-sato`	An image with Sato, a mobile environment and visual style for mobile devices. The image supports a Sato theme and uses Pimlico applications. It contains a terminal, an editor and a file manager.	`poky`
`imx-image-core`	An i.MX image with i.MX test applications to be used for Wayland backends. This image is used by our daily core testing.	`meta-imx/meta-sdk`

Table 1. i.MX Yocto project images...*continued*

Image name	Target	Provided by layer
`fsl-image-machine-test`	An FSL Community i.MX core image with console environment - no GUI interface.	`meta-freescale-distro`
`imx-image-multimedia`	Builds an i.MX image with a GUI without any Qt content.	`meta-imx/meta-sdk`
`imx-image-full`	Builds an opensource Qt 6 image with Machine Learning features. These images are only supported for i.MX SoC with hardware graphics. They are not supported on the i.MX 6UltraLite, i.MX 6UltraLiteLite, i.MX 6SLL, i.MX 7Dual, i.MX 8MNanoLite, or i.MX 8DXL	`meta-imx/meta-sdk`

5.3 Building an image

The Yocto Project build uses the `bitbake` command. For example, `bitbake <component>` builds the named component. Each component build has multiple tasks, such as fetching, configuration, compilation, packaging, and deploying to the target rootfs. The `bitbake` image build gathers all the components required by the image and build in order of the dependency per task. The first build is the toolchain along with the tools required for the components to build.

The following command is an example on how to build an image:

```
$ bitbake imx-image-multimedia
```

5.4 Bitbake options

The `bitbake` command used to build an image is `bitbake <image name>`. Additional parameters can be used for specific activities described below. Bitbake provides various useful options for developing a single component. To run with a BitBake parameter, the command looks like this:

```
bitbake  <parameter> <component>
```

`<component>` is a desired build package.

The following table provides some BitBake options.

Table 2. BitBake options

BitBake parameter	Description
`-c fetch`	Fetches if the downloads state is not marked as done.
`-c cleanall`	Cleans the entire component build directory. All the changes in the build directory are lost. The rootfs and state of the component are also cleared. The component is also removed from the download directory.
`-c deploy`	Deploys an image or component to the rootfs.
`-k`	Continues building components even if a build break occurs.
`-c compile -f`	It is not recommended that the source code under the temporary directory is changed directly, but if it is, the Yocto Project might not rebuild it unless this option is used. Use this option to force a recompile after the image is deployed.
`-g`	Lists a dependency tree for an image or component.
`-DDD`	Turns on debug 3 levels deep. Each D adds another level of debug.
`-s, --show-versions`	Shows the current and preferred versions of all recipes.

5.5 U-Boot configuration

U-Boot configurations are defined in the main machine configuration file. The configuration is specified by using the `UBOOT_CONFIG` settings. This requires setting `UBOOT_CONFIG` in `local.conf`. Otherwise, the U-Boot build uses SD boot by default.

These can be built separately by using the following commands (change `MACHINE` to the correct target). Multiple U-Boot configurations can be built with one command by putting spaces between U-Boot configurations.

The following are the U-Boot configurations for each boards. i.MX 6 and i.MX 7 boards support SD without OP-TEE and with OP-TEE:

- `uboot_config_imx95evk="sd fspi"`
- `uboot_config_imx93evk="sd fspi"`
- `uboot_config_imx8mpevk="sd fspi ecc"`
- `uboot_config_imx8mnevk="sd fspi"`
- `uboot_config_imx8mmevk="sd fspi"`
- `uboot_config_imx8mqevk="sd"`
- `uboot_config_imx8dxlevk="sd fspi"`
- `uboot_conifg_imx8dxmek="sd fspi"`
- `uboot_config_imx8qxpc0mek="sd fspi"`
- `uboot_config_imx8qxpmek="sd fspi"`
- `uboot_config_imx8qmmek="sd fspi"`
- `uboot_config_imx8ulpevk="sd fspi"`
- `uboot_config_imx8ulp-9x9-lpddr4-evk="sd fspi"`
- `uboot_config_imx6qsabresd="sd sata sd-optee"`
- `uboot_config_imx6qsabreauto="sd sata eimnor spinor nand sd-optee"`
- `uboot_config_imx6dlsabresd="sd epdc sd-optee"`
- `uboot_config_imx6dlsabreauto="sd eimnor spinor nand sd-optee"`
- `uboot_config_imx6solosabresd="sd sd-optee"`
- `uboot_config_imx6solosabreauto="sd eimnor spinor nand sd-optee"`
- `uboot_config_imx6sxsabresd="sd emmc qspi2 m4fastup sd-optee"`
- `uboot_config_imx6sxsabreauto="sd qspi1 nand sd-optee"`
- `uboot_config_imx6qpsabreauto="sd sata eimnor spinor nand sd-optee"`
- `uboot_config_imx6qpsabresd="sd sata sd-optee"`
- `uboot_config_imx6sllevk="sd epdc sd-optee"`
- `uboot_config_imx6ulevk="sd emmc qspi1 sd-optee"`
- `uboot_config_imx6ul9x9evk="sd qspi1 sd-optee"`
- `uboot_config_imx6ull14x14evk="sd emmc qspi1 nand sd-optee"`
- `uboot_config_imx6ull9x9evk="sd qspi1 sd-optee"`
- `uboot_config_imx6ulz14x14evk="sd emmc qspi1 nand sd-optee"`
- `uboot_config_imx7dsabresd="sd epdc qspi1 nand sd-optee"`
- `uboot_config_imx7ulpevk="sd emmc sd-optee"`

To build with any U-Boot configuration, perform the following steps.

With just one U-Boot configuration:

```
$ echo "UBOOT_CONFIG = \"eimnor\"" >> conf/local.conf
```

With multiple U-Boot configurations:

```
$ echo "UBOOT_CONFIG = \"sd eimnor\"" >> conf/local.conf
```

```
$ MACHINE=<machine name> bitbake -c deploy u-boot-imx
```

Note: *i.MX 8 uses `imx-boot` that pulls in U-Boot.*

5.6 Build scenarios

The following are build setup scenarios for various configurations.

Set up the manifest and populate the Yocto Project layer sources with these commands:

```
$ mkdir imx-yocto-bsp
$ cd imx-yocto-bsp
$ repo init -u https://github.com/nxp-imx/imx-manifest
-b imx-linux-scarthgap -m imx-6.6.36-2.1.0.xml
$ repo sync
```

The following sections give some specific examples. Replace the machine names and the backends specified to customize the commands.

5.6.1 Frame Buffer image on i.MX 6QuadPlus SABRE-AI

```
$ DISTRO=fsl-imx-fb MACHINE=imx6qpsabresd source imx-setup-release.sh -b build-
fb
$ bitbake imx-image-multimedia
```

This builds a multimedia image with a frame buffer backend.

5.6.2 XWayland image on i.MX 8QuadXPlus MEK

```
$ DISTRO=fsl-imx-xwayland MACHINE=imx8qxpmek source imx-setup-release.sh -b
 build-xwayland
$ bitbake imx-image-full
```

This builds an XWayland image with Qt 6 and machine learning features. To build without Qt 6 and machine learning, use `imx-image-multimedia` instead.

5.6.3 Wayland image on i.MX 8M Quad EVK

```
$ DISTRO=fsl-imx-wayland MACHINE=imx8mqevk source imx-setup-release.sh -b build-
wayland
$ bitbake imx-image-multimedia
```

This builds a Weston Wayland image with multimedia without Qt 6.

5.6.4 Restarting a build environment

If a new terminal window is opened or the machine is rebooted after a build directory is set up, the setup environment script should be used to set up the environment variables and run a build again. The full `imx-setup-release.sh` is not needed.

```
$ source setup-environment <build-dir>
```

5.6.5 Chromium Browser on XWayland, and Wayland

The Yocto Project community has Chromium recipes for the Wayland version Chromium Browser for i.MX SoC with GPU hardware. NXP does not support or test the patches from the community. This section describes how to integrate Chromium into your rootfs and enable hardware accelerated rendering of WebGL. The Chromium browser requires additional layers such as meta-browser added in the `imx-release-setup.sh` script automatically.

In `local.conf` for XWayland or Wayland, add Chromium into your image. X11 is not supported.

```
CORE_IMAGE_EXTRA_INSTALL += "chromium-ozone-wayland"
```

5.6.6 Qt 6 and QtWebEngine browsers

Qt 6 has both a commercial and an open source license. When building in Yocto Project, the open source license is the default. Make sure to understand the differences between these licenses and choose appropriately. After custom Qt 6 development has started on the open source license, it cannot be used with the commercial license. Work with a legal representative to understand the differences between these licenses.

Note:

Building QtWebEngine is not compatible with the meta-chromium layer used by the release.

If you are using the NXP build setup, remove meta-chromium from `bblayers.conf`:

```
# Commented out due to incompatibility with qtwebengine
#BBLAYERS += "${BSPDIR}/sources/meta-browser/meta-chromium"
```

There are four Qt 6 browsers available. QtWebEngine browsers can be found in:

- `/usr/share/qt6/examples/webenginewidgets/StyleSheetbrowser`
- `/usr/share/qt6/examples/webenginewidgets/Simplebrowser`
- `/usr/share/qt6/examples/webenginewidgets/Cookiebrowser`
- `/usr/share/qt6/examples/webengine/quicknanobrowser`

All three browsers can be run by going to the directory above and running the executable found there. Touchscreen can be enabled by adding the parameters `-plugin evdevtouch:/dev/input/event0` to the executable.

```
./quicknanobrowser -plugin evdevtouch:/dev/input/event0
```

QtWebengine only works on SoC with GPU graphics hardware on i.MX 6, i.MX 7, i.MX 8, and i.MX 9.

To include Qtwebengine in the image, put the following in `local.conf` or in the image recipe.

```
IMAGE_INSTALL:append = " packagegroup-qt6-webengine"
```

5.6.7 NXP eIQ machine learning

The `meta-ml` layer is the integration of NXP eIQ machine learning, which was formerly released as a separate `meta-imx-machinelearning` layer and is now integrated into the standard BSP image (`imx-image-full`). Many of the features require Qt 6. In case of using other configuration than `imx-image-full`, put the following in `local.conf`:

```
IMAGE_INSTALL:append = " packagegroup-imx-ml"
```

To install the NXP eIQ packages to the SDK, put the following in `local.conf`:

```
TOOLCHAIN_TARGET_TASK:append = " tensorflow-lite-dev onnxruntime-dev"
```

Note:

`TOOLCHAIN_TARGET_TASK_append` variable installs the packages to the SDK only, not to the image.

To add the model configurations and input data for the OpenCV DNN demos, put the following in `local.conf`:

```
PACKAGECONFIG:append:pn-opencv_mx8 = " tests tests-imx"
```

5.6.8 Systemd

Systemd is enabled as the default initialization manager. To disable systemd as default, go to the `fsl-imx-preferred-env.inc` and comment out the systemd section.

5.6.9 Multilib enablement

For i.MX 8, building 32-bit applications on 64-bit OS can be supported using the multilib configuration. Multilib offers the ability to build libraries with different target optimizations or architecture formats and combine these together into one system image. Multilib is enabled by adding the `MULTILIB`, `DEFAULTTUNE`, and `IMAGE_INSTALL` declaration to your `local.conf` file. Multilib is not supported with the debian package management. It requires the RPM system. Comment out the two package management lines in `local.conf` to go to the default RPM.

The MULTILIBS declaration is typically lib32 or lib64 and needs to be defined in `MULTILIB_GLOBAL_VARIANTS` variable as follows:

```
MULTILIBS = "multilib:lib32"
```

`DEFAULTTUNE` must be one of the `AVAILTUNES` values for this alternative library type as follows:

```
DEFAULTTUNE:virtclass-multilib-lib32 = "armv7athf-neon"
```

`IMAGE_INSTALL` will be added to the image, the 32-bit libraries required by the specific application as follows:

```
IMAGE_INSTALL:append = " lib32-bash"
```

For the case on i.MX 8, building a 32-bit application support would require the following statements in `local.conf`. This configuration specifies a 64-bit machine as the main machine type and adds `multilib:lib32`, where those libraries are compiled with the `armv7athf-neon` tune, and then includes to all images the lib32 packages.

```
MACHINE = imx8mqevk
# Define multilib target
```

```
require conf/multilib.conf
MULTILIBS = "multilib:lib32"
DEFAULTTUNE:virtclass-multilib-lib32 = "armv7athf-neon"
# Add the multilib packages to the image
IMAGE_INSTALL:append = " lib32-glibc lib32-libgcc lib32-libstdc++"
```

Disable the deb packaging to avoid any processing errors. Check in `local.conf`, and comment if there are:

```
PACKAGE_CLASSES = "package_deb"
EXTRA_IMAGE_FEATURES += "package-management"
```

5.6.10 OP-TEE enablement

OP-TEE requires three components: OP-TEE OS, OP-TEE client, and OP-TEE test. In addition, the kernel and U-Boot have configurations. The OP-TEE OS resides in the bootloader while the OP-TEE client and test reside in the rootfs.

OP-TEE is enabled by default in this release. To disable OP-TEE, go to the `meta-imx/meta-imx-bsp/conf/layer.conf` file and comment out the `DISTRO_FEATURES_append` for OP-TEE and uncomment the removed line.

5.6.11 Building Jailhouse

Jailhouse is a static partitioning Hypervisor based on Linux OS. It is supported on i.MX 8M Plus, i.MX 8M Nano, i.MX 8M Quad EVK, and i.MX 8M Mini EVK boards.

To enable Jailhouse build, add the following line to `local.conf`:

```
DISTRO_FEATURES:append = " jailhouse"
```

In U-Boot, run `run jh_netboot` or `jh_mmcboot`. It loads the dedicated DTB for Jailhouse usage. Taking i.MX 8M Quad as an example, after Linux OS boots up:

```
#insmod jailhouse.ko
#./jailhouse enable imx8mq.cell
```

For more details about Jailhouse on i.MX 8, see the *i.MX Linux User's Guide* (UG10163).

5.6.12 Package management

The default package management with Yocto Project is rpm. The i.MX distro now enables debian as the package management. This can be easily turned off by adding the `PACKAGE_CLASSES` set to `package_rpm` in the `local.conf`, or creating a custom distro without the debian package feed `PACKAGE_CLASSES = "package_deb"` .

With the addition of the debian package feed, a `sources.list` can be added to `/etc/apt` that links in Debian's package feed. This allows users to install packages not provided in the image without having to add them to a Yocto image. Because this package feed is not generated by the i.MX Yocto build process, there is no guarantee each package will work with the right dependencies but it allows simpler tools to be provided. Software that is complex and has more dependencies on specific versions might have issues with an external package feed.

6 Image Deployment

Complete filesystem images are deployed to `<build directory>/tmp/deploy/images`. An image is, for the most part, specific to the machine set in the environment setup. Each image build creates a U-Boot, a kernel, and an image type based on the `IMAGE_FSTYPES` defined in the machine configuration file. Most machine configurations provide an SD card image (.wic) and a rootfs image (.tar). The SD card image contains a partitioned image (with U-Boot, kernel, rootfs, etc.) suitable for booting the corresponding hardware.

6.1 Flashing an SD card image

An SD card image file `.wic` contains a partitioned image (with U-Boot, kernel, rootfs, etc.) suitable for booting the corresponding hardware. To flash an SD card image, run the following command:

```
zstdcat <image_name>.wic.zst | sudo dd of=/dev/sd<partition> bs=1M conv=fsync
```

For more information on flashing, see Section "Preparing an SD/MMC card to boot" in the *i.MX Linux User's Guide* (UG10163). For NXP eIQ machine learning applications, an additional free disk space is required (approximately 1 GB). It is defined by adding `IMAGE_ROOTFS_EXTRA_SPACE` variable into the `local.conf` file before the Yocto building process. See the Yocto Project Mega-Manual.

7 Customization

There are three scenarios to build and customize on i.MX Linux OS:

- Building i.MX Yocto Project BSP and validating on an i.MX reference board. The directions in this document describe this method in details.
- Customizing kernel and creating a custom board and device tree with kernel and U-Boot. For more details on how to build an SDK and set up a host machine for building the kernel and U-Boot only outside of the Yocto Project build environment, see Chapter "How to Build U-Boot and Kernel in Standalone Environment" in the *i.MX User's Guide* (UG10163).
- Customizing a distribution adding or removing packaging from the BSP provided for i.MX Linux releases by creating a custom Yocto Project layer. i.MX provides multiple demo examples to show a custom layer on top of an i.MX BSP release . The remaining sections in this document provides instructions for creating a custom DISTRO and board configuration.

7.1 Creating a custom distro

A custom distro can configure a custom build environment. The distro files released `fsl-imx-wayland`, `fsl-imx-xwayland`, and `fsl-imx-fb` all show configurations for specific graphical backends. Distros can also be used to configure other parameters such as kernel, U-Boot, and GStreamer. The i.MX distro files are set to create a custom build environment required for testing our i.MX Linux OS BSP releases.

It is recommended for each customer to create their own distro file and use that for setting providers, versions, and custom configurations for their build environment. A distro is created by copying an existing distro file, or including one like `poky.conf` and adding additional changes, or including one of the i.MX distros and using that as a starting point.

7.2 Creating a custom board configuration

Vendors who are developing reference boards may want to add their board to the FSL Community BSP. Having the new machine supported by the FSL Community BSP makes it easy to share source code with the community, and allows for feedback from the community.

The Yocto Project makes it easy to create and share a BSP for a new i.MX based board. The upstreaming process should start when a Linux OS kernel and a bootloader are working and tested for that machine. It is highly important to have a stable Linux kernel and bootloader (for example, U-Boot) to be pointed to in the machine configuration file, to be the default one used for that machine.

Another important step is to determinate a maintainer for the new machine. The maintainer is the one responsible for keeping the set of main packages working for that board. The machine maintainer should keep the kernel and bootloader updated, and the user-space packages tested for that machine.

The steps needed are listed below.

1. Customize the kernel config files as needed. The kernel configuration file is location in `arch/arm/configs` and the vendor kernel recipe should customize a version loaded through the kernel recipe.
2. Customize U-Boot as needed. See the *i.MX BSP Porting Guide* (IMXBSPPG) for details on this.
3. Assign the maintainer of the board. This maintainer makes sure that files are updated as needed, so the build always works.
4. Set up the Yocto Project build as described in the Yocto Project community instructions as shown below. Use the community master branch.
   a. Download the needed host package, depending on your host Linux OS distribution, from Yocto Project Quick Start.
   b. Download Repo with the command:

   ```
   $ curl https://storage.googleapis.com/git-repo-downloads/repo > ~/bin/repo
   ```

   c. Create a directory to keep everything in. Any directory name can be used. This document uses `imx-community-bsp`.

   ```
   $ mkdir imx-community-bsp
   ```

   d. Execute the following command:

   ```
   $ cd imx-community-bsp
   ```

   e. Initialize the Repo with the master branch of the Repo.

   ```
   $ repo init -u https://github.com/Freescale/fsl-community-bsp-platform -b
    master
   ```

   f. Get the recipes that will be used to build.

   ```
   $ repo sync
   ```

   g. Set up the environment with the following command:

   ```
   $ source setup-environment build
   ```

5. Choose a similar machine file in `fsl-community-bsp/sources/meta-freescale-3rdparty/conf/machine` and copy it, using a name indicative of your board. Edit the new board file with the information about your board. Change the name and description at least. Add `MACHINE_FEATURE`.
6. Test your changes with the latest community master branch, making sure everything works well. Use at least `core-image-minimal`.

   ```
   $ bitbake core-image-minimal
   ```

7. Prepare the patches. Follow the Recipe Style Guide and git.yoctoproject.org/cgit/cgit.cgi/meta-freescale/tree/README in the section entitled *Contributing*.
8. Upstream into `meta-freescale-3rdparty`. To upstream, send the patches to meta-freescale@yoctoproject.org.

7.3 Monitoring security vulnerabilities in your BSP

The monitoring of Common Vulnerability and Exposures (CVE) can be accomplished with NXP enabled Vigiles tools from Timesys. Vigiles is a vulnerability monitoring and management tool that provides build-time Yocto CVE analysis of target images. It does this by collecting metadata about the software used in the Yocto Project BSP and comparing it against a CVE database that integrates information on CVEs from various sources, including NIST, Ubuntu, and several others.

A high-level overview of the detected vulnerabilities is returned, and a full detailed analysis with information on affecting CVEs, their severity and available fixes can be viewed online.

To access the report online, register for your NXP Vigiles account by following the link:

https://www.timesys.com/register-nxp-vigiles/

Additional information on the setup and execution of Vigiles can be found here:

https://github.com/TimesysGit/meta-timesys

https://www.nxp.com/vigiles

7.3.1 Configuration

Add `meta-timesys` to `conf/bblayers.conf` of your BSP build.

Follow the format of the file and add meta-timesys:

```
BBLAYERS += "${BSPDIR}/sources/meta-timesys"
```

Append vigiles to `INHERIT` variable in `conf/local.conf`:

```
INHERIT += "vigiles"
```

7.3.2 Execution

Once meta-timesys has been added to your build, Vigiles executes a security vulnerabilities scan every time the Linux BSP is built with Yocto. There are no additional commands needed. After each build is completed, the vulnerability scan information is stored in the directory `imx-yocto-bsp/<build dir>/vigiles`.

You can view the details of the security scan through:

- Command line (summary)
- Online (details)

Simply open the file named `<image name>-report.txt`, which includes the link to the detailed online report.

8 Frequently Asked Questions

8.1 Quick Start

This section summarizes how to set up the Yocto Project on a Linux machine and build an image. Detailed explanations of what this means are in the sections above.

Installing the "repo" utility

To get the BSP you need to have "repo" installed. This only needs to be done once.

```
$: mkdir ~/bin
```

```
$: curl https://storage.googleapis.com/git-repo-downloads/repo > ~/bin/repo
$: chmod a+x ~/bin/repo
$: PATH=${PATH}:~/bin
```

Downloading the BSP Yocto Project Environment.

Use the correct name for the release desired in the `-b` option for `repo init`. This needs to be done once for each release and sets the distribution for the directory created in the first step. `repo sync` can be run to update the recipes under `sources` to the latest.

```
$: mkdir imx-yocto-bsp
$: cd imx-yocto-bsp
$: repo init -u  https://github.com/nxp-imx/imx-manifest -b imx-linux-scarthgap
 -m imx-6.6.36-2.1.0.xml
: repo sync
```

Note:

https://github.com/nxp-imx/imx-manifest/tree/imx-linux-scarthgap has a list of all manifest files supported in this release.

Setup for Specific Backends

i.MX 8 and i.MX 9 Framebuffer is not supported. Only use these for i.MX 6 and i.MX 7 SoC.

Setup for Framebuffer:

```
$: DISTRO=fsl-imx-fb MACHINE=<machine name> source imx-setup-release.sh -b
 build-fb
```

Setup for Wayland:

```
$: DISTRO=fsl-imx-wayland MACHINE=<machine name> source imx-setup-release.sh -b
 build-wayland
```

Setup for XWayland:

```
$: DISTRO=fsl-imx-xwayland MACHINE=<machine name> source imx-setup-release.sh -b
 build-xwayland
```

Build for All Backends

Build without Qt

```
$: bitbake imx-image-multimedia
```

Build with Qt 6 and machine learning features

```
$: bitbake imx-image-full
```

8.2 Local configuration tuning

A Yocto Project build can take considerable build resources both in time and disk usage, especially when building in multiple build directories. There are methods to optimize this, for example, use a shared sstate cache

(caches the state of the build) and downloads directory (holds the downloaded packages). These can be set to be at any location in the `local.conf` file by adding statements such as these:

```
DL_DIR="/opt/imx/yocto/imx/download"
SSTATE_DIR="/opt/imx/yocto/imx/sstate-cache"
```

The directories need to already exist and have appropriate permissions. The shared sstate helps when multiple build directories are set, each of which uses a shared cache to minimize the build time. A shared download directory minimizes the fetch time. Without these settings, Yocto Project defaults to the build directory for the sstate cache and downloads.

Every package downloaded in the `DL_DIR` directory is marked with a `<package name>.done`. If your network has a problem fetching a package, you can manually copy the backup version of package to the `DL_DIR` directory and create a `<package_name>.done` file with the touch command. Then run the bitbake command: `bitbake <component>`.

For more information, see the Yocto Project Reference Manual.

8.3 Recipes

Each component is built by using a recipe. For new components, a recipe must be created to point to the source (`SRC_URI`) and specify patches, if applicable. The Yocto Project environment builds from a makefile in the location specified by the `SRC_URI` in the recipe. When a build is established from auto tools, a recipe should inherit `autotools` and `pkgconfig`. Makefiles must allow CC to be overridden by Cross Compile tools to get the package built with Yocto Project.

Some components have recipes but need additional patches or updates. This can be done by using a bbappend recipe. This appends to an existing recipe details about the updated source. For example, a bbappend recipe to include a new patch should have the following contents:

```
FILESEXTRAPATHS:prepend := "${THISDIR}/${PN}:"
SRC_URI += file://<patch name>.patch
```

`FILESEXTRAPATHS_prepend` tells Yocto Project to look in the directory listed to find the patch listed in `SRC_URI`.

Note:

If a `bbappend` recipe is not picked up, view the fetch log file (`log.do_fetch`) under the work folder to check whether the related patches are included or not. Sometimes a Git version of the recipe is being used instead of the version in the `bbappend` files.

8.4 How to select additional packages

Additional packages can be added to images if there is a recipe provided for that package. A searchable list of recipes provided by the community can be found at layers.openembedded.org/. You can search to see if an application already has a Yocto Project recipe and find where to download it from.

8.4.1 Updating an image

An image is a set of packages and the environment configuration.

An image file (such as `imx-image-multimedia.bb`) defines the packages that go inside the file system. Root file systems, kernels, modules, and the U-Boot binary are available in `build/tmp/deploy/images/<machine name>`.

Note:

You can build packages without including it in an image, but you must rebuild the image if you want the package installed automatically on a rootfs.

8.4.2 Package group

A package group is a set of packages that can be included on any image.

A package group can contain a set of packages. For example, a multimedia task could determine, according to the machine, whether the VPU package is built or not, so the selection of multimedia packages may be automated for every board supported by the BSP, and only the multimedia package is included in the image.

Additional packages can be installed by adding the following line in `<build dir>/local.conf`.

```
CORE_IMAGE_EXTRA_INSTALL:append = " <package_name1 package_name2>"
```

There are many package groups. They are in subdirectories named `packagegroup` or `packagegroups`.

8.4.3 Preferred version

The preferred version is used to specify the preferred version of a recipe to use for a specific component. A component may have multiple recipes in different layers and a preferred version points to a specific version to use.

In the `meta-imx` layer, in `layer.conf`, preferred versions are set for all the recipes to provide a static system for a production environment. These preferred version settings are used for formal i.MX releases but are not essential for future development.

Preferred versions also help when previous versions may cause confusion about which recipe should be used. For example, previous recipes for `imx-test` and `imx-lib` used a year-month versioning, which has changed to `<kernel-version>` versioning. Without a preferred version, an older version might be picked up. Recipes that have `_git` versions are usually picked over other recipes, unless a preferred version is set. To set a preferred version, put the following in `local.conf`.

```
PREFERRED_VERSION_<component>:<soc family> = "<version>"
```

See the Yocto Project manuals for more information on using preferred versions.

8.4.4 Preferred provider

The preferred provider is used to specify the preferred provider for a specific component. A component can have multiple providers. For example, the Linux kernel can be provided by i.MX or by `kernel.org` and preferred provider states the provider to use.

For example, U-Boot is provided by both the community through `denx.de` and i.MX. The community provider is specified by `u-boot-fslc`. The i.MX provider is specified by `u-boot-imx`. To state a preferred provider, put the following in `local.conf`:

```
PREFERRED_PROVIDER_<component>:<soc family> = "<provider>"
PREFERRED_PROVIDER_u-boot_mx6 = "u-boot-imx"
```

8.4.5 SoC family

The SoC family documents a class of changes that apply to a specific set of system chips. In each machine configuration file, the machine is listed with a specific SoC family. For example, i.MX 6DualLite Sabre-SD is listed under the i.MX 6 and i.MX 6DualLite SoC families. i.MX 6Solo Sabre-auto is listed under the i.MX 6 and

i.MX 6Solo SoC families. Some changes can be targeted to a specific SoC family in `local.conf` to override a change in a machine configuration file. The following is an example of a change to an mx6dlsabresd kernel setting.

```
KERNEL_DEVICETREE:mx6dl = "imx6dl-sabresd.dts"
```

SoC families are useful when making a change that is specific only for a class of hardware. For example, i.MX 28 EVK does not have a Video Processing Unit (VPU), so all the settings for VPU should use i.MX 5 or i.MX 6 to be specific to the correct class of chips.

8.4.6 BitBake logs

BitBake logs the build and package processes in the temp directory in `tmp/work/<architecture>/<component>/temp`.

If a component fails to fetch a package, the log showing the errors is in the file `log.do_fetch`.

If a component fails to compile, the log showing the errors is in the file `log.do_compile`.

Sometimes a component does not deploy as expected. Check the directories under the build component directory (`tmp/work/<architecture>/<component>`). Check the package, `packages-split`, and `sysroot*` directories of each recipe to see if the files are placed there (where they are staged prior to being copied to the deploy directory).

8.4.7 How to add a mechanism for CVE monitoring and notification

The CVE tracking mechanism can be fetched from GitHub. Navigate to the directory `imx-yocto-bsp/sources`.

Run the following command:

```
git clone https://github.com/TimesysGit/meta-timesys.git -b scarthgap
```

This command will download an additional metalayer that provides scripts for image manifest generation used for security monitoring and notification as part of the Vigiles product offering from NXP and Timesys. Follow Section 7.3 on how to use the solution.

Getting access to full CVE reporting requires a LinuxLink License Key. Without the key in your development environment, Vigiles continues to execute in Demo Mode, producing summary reports only.

Log into your Vigiles account on LinuxLink (or create one if you do not have: https://www.timesys.com/register-nxp-vigiles/). Access your Preferences and generate a New Key. Download the key file to your development environment. Specify the location of the key file in your Yocto's `conf/local.conf` file with the following statement:

```
VIGILES_KEY_FILE = "/tools/timesys/linuxlink_key"
```

9 References

- For details on boot switches, see Section "How to Boot the i.MX Boards" in the *i.MX Linux User's Guide* (UG10163).
- For how to download images using U-Boot, see Section "Downloading Images Using U-Boot" in the *i.MX Linux User's Guide* (UG10163).
- For how to set up an SD/MMC card, see Section "Preparing an SD/MMC Card to Boot" in the *i.MX Linux User's Guide* (UG10163).

10 Note About the Source Code in the Document

Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2024 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

11 Revision History

This table provides the revision history.

Revision history

Document ID	Date	Substantive changes
UG10164 v.LF6.6.36_2.1.0	30 September 2024	Upgraded to the 6.6.36 kernel.
IMXLXYOCTOUG_6.6.23_2.0.0	4 July 2024	Corrected a typo in the command lines in Section 4.
IMXLXYOCTOUG_6.6.23_2.0.0	28 June 2024	Upgraded to the 6.6.23 kernel, U-Boot v2024.04, TF-A v2.10, OP-TEE 4.2.0, Yocto 5.0 Scarthgap, and added the i.MX 91 as Alpha quality, i.MX 95 as Beta quality.
IMXLXYOCTOUG v.LF6.6.3_1.0.0	29 March 2024	Upgraded to the 6.6.3 kernel, removed the i.MX 91P, and added the i.MX 95 as Alpha Quality.
IMXLXYOCTOUG v.LF6.1.55_2.2.0	12/2023	Upgraded to the 6.1.55 kernel.
IMXLXYOCTOUG v.LF6.1.36_2.1.0	09/2023	Upgraded to the 6.1.36 kernel and added the i.MX 91P.
IMXLXYOCTOUG v.LF6.1.22_2.0.0	06/2023	Upgraded to the 6.1.22 kernel.
IMXLXYOCTOUG v.LF6.1.1_1.0.0	04/2023	Error correction to the command lines in Section 3.2.
IMXLXYOCTOUG v.LF6.1.1_1.0.0	03/2023	Upgraded to the 6.1.1 kernel.
IMXLXYOCTOUG v.LF5.15.71_2.2.0	12/2022	Upgraded to the 5.15.71 kernel.
IMXLXYOCTOUG v.LF5.15.52_2.1.0	09/2022	Upgraded to the 5.15.52 kernel, and added the i.MX 93.
IMXLXYOCTOUG v.LF5.15.32_2.0.0	06/2022	Upgraded to the 5.15.32 kernel, U-Boot 2022.04, and Kirkstone Yocto.
IMXLXYOCTOUG v.LF5.15.5_1.0.0	03/2022	Upgraded to the 5.15.5 kernel, Honister Yocto, and Qt6.

Revision history...*continued*

Document ID	Date	Substantive changes
IMXLXYOCTOUG v.LF5.10.72_2.2.0	12/2021	Upgraded the kernel to 5.10.72 and updated the BSP.
IMXLXYOCTOUG v.LF5.10.52_2.1.0	09/2021	Updated for i.MX 8ULP Alpha and the kernel upgraded to 5.10.52.
IMXLXYOCTOUG v.LF5.10.35_2.0.0	06/2021	Upgraded to 5.10.35 kernel.
IMXLXYOCTOUG v.LF5.10.9_1.0.0	04/2021	Corrected a typo in the command lines in Section 3.1 "Host packages".
IMXLXYOCTOUG v.LF5.10.9_1.0.0	03/2021	Upgraded to 5.10.9 kernel.
IMXLXYOCTOUG v.L5.4.70_2.3.0	01/2021	Updated the command lines in Section "Running the Arm Cortex-M4 image".
IMXLXYOCTOUG v.L5.4.70_2.3.0	12/2020	i.MX 5.4 consolidated GA for release i.MX boards including i.MX 8M Plus and i.MX 8DXL.
IMXLXYOCTOUG v.L5.4.47_2.2.0	09/2020	i.MX 5.4 Beta2 release for i.MX 8M Plus, Beta for 8DXL, and consolidated GA for released i.MX boards.
IMXLXYOCTOUG v.L5.4.24_2.1.0	06/2020	i.MX 5.4 Beta release for i.MX 8M Plus, Alpha2 for 8DXL, and consolidated GA for released i.MX boards.
IMXLXYOCTOUG v.L5.4.3_2.0.0	04/2020	i.MX 5.4 Alpha release for i.MX 8M Plus and 8DXL EVK boards.
IMXLXYOCTOUG v.LF5.4.3_1.0.0	03/2020	i.MX 5.4 Kernel and Yocto Project Upgrades.
IMXLXYOCTOUG v.L4.19.35_1.1.0	10/2019	i.MX 4.19 Kernel and Yocto Project Upgrades.
IMXLXYOCTOUG v.L4.19.35_1.0.0	07/2019	i.MX 4.19 Beta Kernel and Yocto Project Upgrades.
IMXLXYOCTOUG v.L4.14.98_2.0.0_ga	04/2019	i.MX 4.14 Kernel upgrade and board updates.
IMXLXYOCTOUG v.L4.14.78_1.0.0_ga	01/2019	i.MX 6, i.MX 7, i.MX 8 family GA release.
IMXLXYOCTOUG v.L4.14.62_1.0.0_beta	11/2018	i.MX 4.14 Kernel Upgrade, Yocto Project Sumo upgrade.
IMXLXYOCTOUG v.L4.9.123_2.3.0_8mm	09/2018	i.MX 8M Mini GA release.
IMXLXYOCTOUG v.L4.9.88_2.2.0_8qxp-beta2	07/2018	i.MX 8QuadXPlus Beta2 release.
IMXLXYOCTOUG v.L4.9.88_2.1.0_8mm-alpha	06/2018	i.MX 8M Mini Alpha release.
IMXLXYOCTOUG v.L4.9.88_2.0.0-ga	05/2018	i.MX 7ULP and i.MX 8M Quad GA release.
IMXLXYOCTOUG v.L4.9.51_imx8mq-ga	03/2018	Added i.MX 8M Quad GA.
IMXLXYOCTOUG v.L4.9.51_8qm-beta2/8qxp-beta	02/2018	Added i.MX 8QuadMax Beta2 and i.MX 8QuadXPlus Beta.
IMXLXYOCTOUG v.L4.9.51_imx8mq-beta	12/2017	Added i.MX 8M Quad.
IMXLXYOCTOUG v.L4.9.51_imx8qm-beta1	12/2017	Added i.MX 8QuadMax.
IMXLXYOCTOUG v.L4.9.51_imx8qxp-alpha	11/2017	Initial release.

Legal information

Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at https://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this document expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

HTML publications — An HTML version, if available, of this document is provided as a courtesy. Definitive information is contained in the applicable document in PDF format. If there is a discrepancy between the HTML document and the PDF document, the PDF document has priority.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP B.V. — NXP B.V. is not an operating company and it does not distribute or sell products.

Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

Contents